06 | WAT：如何让一个 WebAssembly 二进制模块的内容易于解读?

2020-09-16 于航

WebAssembly入门课 进入课程 >

讲述：于航
时长 10:17 大小 9.43M

你好，我是于航。

在前面的两节课中，我们分别讲解了 Wasm 模块在二进制层面的基本组成结构与数据编码方式。在 04 的结尾，我们还通过一个简单的例子，逐个字节地分析了定义在 C/C++ 源代码中的函数，在被编译到 Wasm 之后所对应的字节码组成结构。

比如字节码 “0x60 0x2 0x7f 0x7f 0x1 0x7f” ，便表示了 Type Section 中定义的一个函数类型（签名）。而该函数类型为 “接受两个 i32 类型参数，并返回一个 i32 类型值

我相信，无论你对 Wasm 的字节码组成结构、V-ISA 指令集中的各种指令使用方式有多么熟悉，在仅通过二进制字节码来分析一个 Wasm 模块时，都会觉得无从入手。那感觉仿佛

是在上古时期时，直接面对着机器码来调试应用程序。那么，有没有一种更为简单、更具有可读性的方式来解读一个 Wasm 模块的内容呢？答案，就在 WAT。

WAT（WebAssembly Text Format）

首先，我们来直观地感受一下 WAT 的“样貌”。假设我们有如下这样一段 C/C++ 源代码，在这段代码中，我们定义了一个函数 factorial，该函数接受一个 int 类型的整数 n，然后返回该整数所对应的阶乘。现在，我们来将它编译成对应的 WAT 代码。

复制代码

```
int factorial(int n) {
  if (n == 0) {
    return 1;
  } else {
    return n * factorial(n-1);
  }
}
```

经过编译和转换后，该函数对应的 WAT 文本代码如下所示。

复制代码

```
(func $factorial (; 0 ;) (param $0 i32) (result i32)
 (local $1 i32)
 (local $2 i32)
 (block $label$0
  (br_if $label$0
   (i32.eqz
    (get_local $0)
   )
  )
  (set_local $2
   (i32.const 1)
  )
  (loop $label$1
   (set_local $2
    (i32.mul
     (get_local $0)
     (get_local $2)
    )
   )
   (set_local $0
    (tee_local $1
     (i32.add
      (get_local $0)
```

```
      (i32.const -1)
     )
    )
   )
   (br_if $label$1
    (get_local $1)
   )
  )
  (return
   (get_local $2)
  )
 )
 (i32.const 1)
)
```

WAT 的全称 “WebAssembly Text Format”，我们一般称其为 “WebAssembly 可读文本格式”。它是一种与 Wasm 字节码格式完全等价，可用于编码 Wasm 模块及其相关定义的文本格式。

这种格式使用 “S- 表达式” 的形式来表达 Wasm 模块及其定义，将组成模块各部分的字节码用一种更加线性的、可读的方式进行表达。

这种文本格式可以被 Wasm 相关的编译工具直接使用，比如 WAVM 虚拟机、Binaryen 调试工具等。不仅如此，Web 浏览器还会在 Wasm 模块没有与之对应的 source-map 数据时（即无法显示模块对应的源语言代码，比如 C/C++ 代码），使用对应的 WAT 可读文本格式代码来作为代替，以方便开发者进行调试。

OK，既然我们之前提到，WAT 使用了 “S- 表达式” 的形式来表达 Wasm 模块及其相关定义，那么接下来，我们就来看看这个 “S- 表达式” 究竟是什么？

S- 表达式（S-Expression）

“S- 表达式”，又被称为 “S-Expression”，或者简写为 “sexpr”，它是一种用于表达树形结构化数据的记号方式。最初，S- 表达式被用于 Lisp 语言，表达其源代码以及所使用到的字面量数据。比如，在 Common Lisp 这个 Lisp 方言中，我们可以有如下形式的一段代码。

复制代码

```
(print
 (* 2 (+ 3 4))
```

```
3 )
```

不知道你有没有感受到，这段 Lisp 代码与之前我们生成的函数 factorial 所对应 WAT 可读文本代码，在结构上有着些许的相似。在这段代码中，我们调用了名为 print 的方法，将一个简单数学表达式 “2 * (3 + 4)” 的计算结果值，打印到了系统的标准输出流（stdout）中。

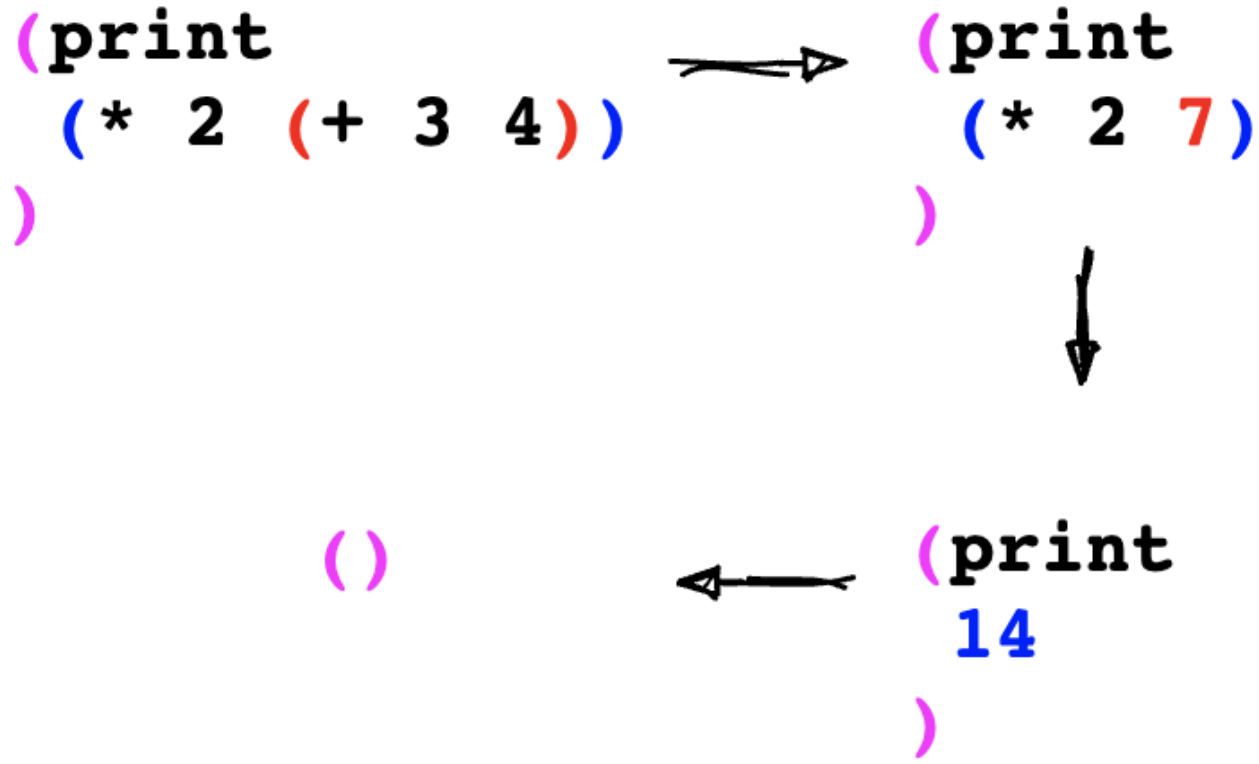
在 “S- 表达式” 中，我们使用一对小括号 “()” 来定义每一个表达式的结构。而表达式之间的相互嵌套关系则表达了一定的语义规则。比如在上面的 Lisp 代码中，子表达式 “(* 2 (+ 3 4))” 的值直接作为了 print 函数的输入参数。而对于这个子表达式本身，也通过内部嵌套的括号表达式及运算符，规定了求值的具体顺序和规则。

不仅如此，每一个表达式在求值时，都会将该表达式将要执行的“操作”，作为括号结构的第一个元素，而对应该操作的具体操作“内容”则紧跟其后。

这里我将“操作”和“内容”都加上了引号，因为 “S- 表达式” 可以被应用于多种不同的场景中，所以这里的操作可能是指一个函数、一个 V-ISA 中的指令，甚至是标识一个结构的标识符。而所对应的“内容”也可以是不同类型的元素或结构。因此，这里你只要了解这种通过括号划分出的所属关系就可以了。

对一个 “S- 表达式” 的求值会从最内层的括号表达式开始。比如对于上述的 Lisp 代码，我们会首先计算其最内层表达式 “(+ 3 4)” 的值。计算完毕后，该括号表达式的位置会由该表达式的计算结果进行替换。以此类推，从内到外，最后计算出整个表达式的值。当然，除了求值，对于诸如 print 函数来说，也会产生一些如“与操作系统 IO 进行交互”之类的副作用（Side Effect）。

你可以参考下面这张图来理解 “S- 表达式” 的组成结构与求值方式（以上述 Lisp 代码为例）。



我们再把目光移回到 WAT 身上。既然我们说，WAT 具有与 Wasm 字节码完全等价的表达能力，可以完全表达通过 Wasm 字节码定义的 Wasm 模块内容。那么从高级语言源代码，到 Wasm 模块字节码、再到对应的 WAT 可读文本代码，这三者是如何做到一一对应的呢?

源码、字节码与 Flat-WAT

为了能够让你更加直观地看清楚从源代码、Wasm 字节码再到 WAT 三者之间的对应关系，首先我们要做的第一件事就是将对应的 WAT 代码 “拍平（flatten）”，将其变成 “Flat-WAT”。这里还是以 “factorial” 函数对应生成的 WAT 可读文本代码为例。

“拍平”的过程十分简单。正常在通过 “S- 表达式” 形式表达的 WAT 代码中，我们通过 “嵌套” 与 “小括号” 的方式指定了各个表达式的求值顺序。而 “拍平” 的过程就是将这些嵌套以及括号结构去掉，以 “从上到下” 的先后顺序，来表达整个程序的执行流程。

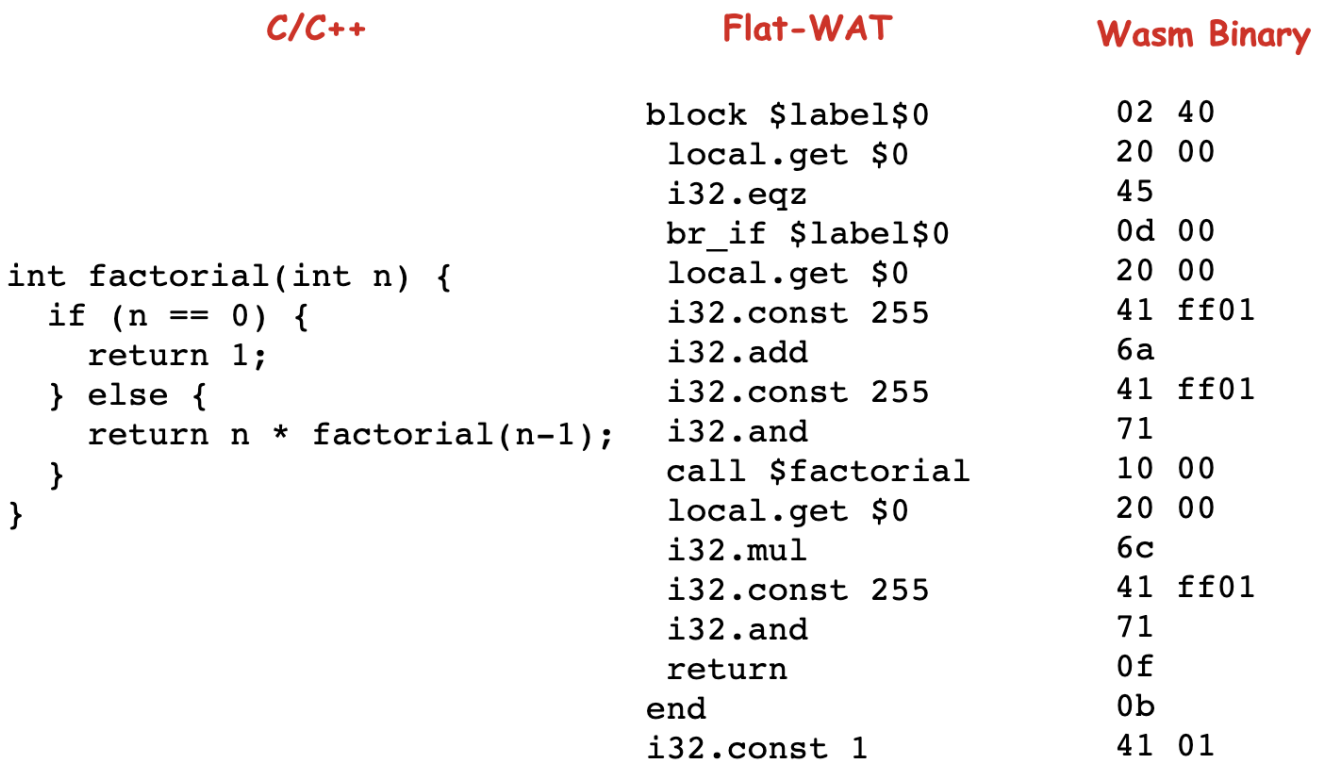
上述 WAT 代码在被 “拍平” 之后，我们可以得到如下所示的 Flat-WAT 代码（这里我们只列出函数体所对应的部分）。

复制代码

```
1 (func $factorial (param $0 i32) (result i32)
2  block $label$0
```

```
  local.get $0
  i32.eqz
  br_if $label$0
  local.get $0
  i32.const 255
  i32.add
  i32.const 255
  i32.and
  call $factorial
  local.get $0
  i32.mul
  i32.const 255
  i32.and
  return
end
i32.const 1)
```

然后我们再将对应 “factorial” 函数的 C/C++ 源代码、Wasm 字节码以及上述 WAT 经过转换生成的 Flat-WAT 代码放到一起，相信你会有一个更加直观的感受。如下图所示，你可以看到 Flat-WAT 代码与 Wasm 字节码会有着直观的“一对一”关系。



模块结构与 WAT

除了我们前面看到的，WAT 可以通过“S- 表达式”的形式，来描述一个定义在 Wasm 模块内的函数定义以外，WAT 还可以描述与 Wasm 模块定义相关的其他部分，比如模块中

各个 Section 的具体结构。如下所示，这是用于构成一个完整 Wasm 模块定义的其他字节码组成部分，所对应的 WAT 可读文本代码。

复制代码

```
(module
 (table 0 anyfunc)
 (memory $0 1)
 (export "memory" (memory $0))
 (export "factorial" (func $factorial))
 ...
)
```

在这里，我们仍然使用 “S- 表达式” 的形式，通过为子表达式指定不同的“操作”关键字，进而赋予每个表达式不同的含义。

比如带有 “table” 关键字的子表达式，定义了 Table Section 的结构。其中的 “0” 表示该 Section 的初始大小为 0，随后紧跟的 “anyfunc” 表示该 Section 可以容纳的元素类型为函数指针类型。其他的诸如 “memory” 表达式定义了 Memory Section，“export” 表达式定义了 Export Section，以此类推。

WAT 与 WAST

在 Wasm 的发展初期，曾出现过一种以 “.wast” 为后缀的文本文件格式，这种文本文件经常被用来存放类似 WAT 的代码内容。

但实际上，以 “.wast” 为后缀的文本文件通常表示着 “.wat” 的一个超集。也就是说，在该文件中可能会包含有一些，基于 WAT 可读文本格式代码标准扩展而来的其他语法结构。比如一些与“断言”和“测试”有关的代码，而这部分语法结构并不属于 Wasm 标准的一部分。

相反的，以 “.wat” 为后缀结尾的文本文件，通常只能够包含有 Wasm 标准语法所对应的 WAT 可读文本代码。并且在一个文本文件中，我们也只能够定义单一的 Wasm 模块结构。

因此，在日常的 Wasm 学习、开发和调试过程中，我更推荐你使用 “.wat” 这个后缀，来作为包含有 WAT 代码的文本文件扩展名。这样可以保障该文件能够具有足够高的兼容

性，能够适配大多数的编译工具，甚至是浏览器来进行识别和解析。

WAT 相关工具

在这节课的最后，我们来看看与 WAT 相关的编译工具。为了使用下面这些工具，你需要安装名为 WABT（The WebAssembly Binary Toolkit）的 Wasm 工具集。关于如何进行安装，你可以在这里找到答案。安装完毕后，我们便可以使用如下这些工具来进行 WAT 代码的相关处理。

wasm2wat：该工具主要用于将指定文件内的 Wasm 二进制代码转译为对应的 WAT 可读文本代码。

wat2wasm：该工具的作用恰好与 wasm2wat 相反。它可以将输入文件内的 WAT 可读文本代码转译为对应的 Wasm 二进制代码。

wat-desugar：该工具主要用于将输入文件内的，基于 “S- 表达式” 形式表达的 WAT 可读文本代码“拍平”成对应的 Flat-WAT 代码。

上述这三个工具的用法十分简单，默认情况下，转译生成的目标代码将被输出到操作系统的标准输出流中。当然，你也可以通过 “-o” 参数来指定输出结果的保存文件。更详细的信息，你可以直接参考该项目在 Github 上的帮助文档。

总结

好了，讲到这，今天的内容也就基本结束了。最后我来给你总结一下。

本节课我们主要讲解了 WAT，这是一种可以将 Wasm 二进制字节码基于 “S- 表达式” 的结构，用“人类可读”的方式展现出来的文本代码格式。

WAT 使用嵌套的“括号表达式”结构来表达 Wasm 字节码的内容，表达式由“操作”关键字与相应的“内容”两部分组成。Wasm 字节码与 WAT 可读文本代码两者之间是完全等价的。

WAT 还有与之相对应的 Flat-WAT 形式的代码。在这个类型的代码中，WAT 内部嵌套的表达式结构（主要是指函数定义部分）将由按顺序平铺开的，由上至下的指令执行结构作为代替。

除此之外，我们还讲解了 “.wast” 与 “.wat” 两种文本文件格式之间的区别。其中，前者为后者的超集，其内部可能会含有与“测试”和“断言”相关的扩展性语法结构；而后者仅包含有与 Wasm 标准相关的可读文本代码结构。因此，在日常编写 WAT 的过程中，建议你以 “.wat” 作为保存 WAT 代码的文本文件后缀。

最后，我们还介绍了几个可以用来与 WAT 格式打交道的工具。这几个工具均来自于名为 WABT 的 Wasm 二进制格式工具集，它们的用法都十分简单，相信你可以快速上手。

课后练习

最后，我们来做一个小练习吧。

尝试使用 C/C++ 编写一个“计算第 n 项斐波那契数列值”的函数 fibonacci，然后在 WasmFiddle 上编译你的函数，并查看对应生成的 WAT 可读文本代码。

今天的课程就结束了，希望可以帮助到你，也希望你在下方的留言区和我参与讨论，同时欢迎你把这节课分享给你的朋友或者同事，一起交流一下。

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 05 | 二进制编码：WebAssembly 微观世界的基本数据规则是什么?

下一篇 07 | WASI：你听说过 WebAssembly 操作系统接口吗?

精选留言 (4)

写留言

慌慌张张
2020-10-20

老师您好，请教一个问题。我们一般都是把c或者c++直接编译成wasm，只要native通了，wasm也没问题。那么出现wat得意义在哪里？貌似不需要通过wat来调试之类的……

作者回复: 其实就如同 WAT 的字面意思，这种格式就是为了能够以更加方便的形式来阅读 Wasm 字节码的内容。因为并不是所有的 Wasm 字节码都有其对应的源代码。对于一些特殊情况，我们可以直接人工编写 WAT 代码然后编译的字节码使用。

1

sprinty

2020-10-02

感觉 Flat-WAT，比 WAT 看着好懂，为啥不直接只用 Flat-WAT?

作者回复: 这个主要是由于对“S-表达式”这种代码表达方式的选择。而为什么选择“S-表达式”则是由于几个因素的考虑：1、尽量不自行创建新的格式，而是直接利用已有的成熟的格式；2、这种格式可以“转换”为方便人们阅读的形式；3、这种格式可以表达 Wasm 模块的内部结构，与字节码一一对应。综上，核心团队选择了“S-表达式”。而对于编译器和工具来说，这种“S-表达式”可以被现有的很对代码实现直接解析和使用，不用重新造轮子，减轻了 Wasm 早期发展时的难度和负担。而同时“S-表达式”也可以被转换为对应的“Linear Representation”的形式，也就是“Flat-WAT”的格式。所以因果关系是先有的“S-表达式”形式的 WAT，才有的对应的 Flat-WAT。

1

军秋

2020-09-26

老师能在浏览器中像调试js一样调试wasm的C代码吗？（浏览器source中显示C代码，断点单步调试？）我自己尝试了一些方法还没成功

展开

作者回复: 可以的，具体可以参考这篇文章：https://developers.google.com/web/updates/2019/12/webassembly。我在后面实践篇的文章中也会介绍哈。

Jason

2020-09-17

老师的原理讲解很细致，受益匪浅，以前自认为自己了解了 wasm，课程看到此意识到自己所知甚少。谢谢于老师的讲解，期待于航老师的实战讲解！

展开

作者回复: 很高兴能够对你有帮助！